华中科技大学

课程报告

课程名称：大数据分析报告—推荐系统

专业班级：大数据 2101 班

学　　号：U202115652

姓　　名：李嘉鹏

指导教师：王蔚

报告日期：2023 年 5 月 15 日

计算机科学与技术学院

目 录

大数据分析大作业 推荐系统

1.1 研究目的及意义

1、了解推荐系统的多种推荐算法并理解其原理。
2、实现 **User-User 的协同过滤算法**并对用户进行推荐。
3、实现**基于内容的推荐算法**并对用户进行推荐。
4、对两个算法进行电影预测评分对比。
5、在学有余力的情况下，**加入 minihash 算法**对效用矩阵**进行降维处理**。

1.2 实验内容

给定 MovieLens 数据集，包含电影评分，电影标签等文件，其中电影评分文件分为训练集 train_set 和测试集 test_set 两部分

基础版必做一：**基于用户的协同过滤推荐算法**

对训练集中的评分数据构造用户-电影效用矩阵，使用 **pearson 相似度计算方法计算用户之间的相似度**，也即相似度矩阵。**对单个用户进行推荐时，找到与其最相似的 k 个用户，用这 k 个用户的评分情况对当前用户的所有未评分电影进行评分预测，选取评分最高的 n 个电影进行推荐。预测评分按照以下方式计算：**

$$\text{predict\_rating} = \frac{\sum_{i=1}^{k} rating(i) * sim(i)}{\sum_{i=1}^{k} sim(i)}$$

在测试集中包含 100 条用户-电影评分记录，用于计算推荐算法中预测评分的准确性，对测试集中的每个用户-电影需要**计算其预测评分，再和真实评分进行对比，误差计算使用 SSE 误差平方和**。

选做部分提示：此算法的进阶版采用 minihash 算法对效用矩阵进行降维处理，从而得到相似度矩阵，注意 minihash 采用 jarcard 方法计算相似度，需要对效用矩阵进行 01 处理，也即**将 0.5-2.5 的评分置为 0，3.0-5.0 的评分置为 1**。

基础版必做二：**基于内容的推荐算法**

将数据集 movies.csv 中的电影类别作为特征值，**计算这些特征值的 tf-idf 值**，得到关于电影与特征值的 n（电影个数）*m（特征值个数）的 **tf-idf 特征矩阵**。根据得到的 tf-idf 特征矩阵，**用余弦相似度的计算方法，得到电影之间的相似度矩阵**。

对某个用户-电影进行预测评分时，获取当前用户的已经完成的所有电影的打分，通过电影相似度矩阵获得已打分电影与当前预测电影的相似度，按照下列方式进行打分计算：

$$\text{score} = \frac{\sum_{i=1}^{n} score'(i) * sim(i)}{\sum_{i=1}^{n} sim(i)}$$

选取相似度大于零的值进行计算，如果已打分电影与当前预测用户-电影相似度大于零，加入计算集合，否则丢弃。（相似度为负数的，强制设置为 0，表示无相关）假设计算集合中一共有 n 个电影，score 为我们预测的计算结果，score'(i)为计算集合中第 i 个电影的分数，sim(i)为第 i 个电影与当前用户-电影的相似度。如果 n 为零，则 score 为该用户所有已打分电影的平均值。

要求能够对指定的 userID 用户进行电影推荐，推荐电影为预测评分排名前 k 的电影。userID 与 k 值可以根据需求做更改。

推荐算法准确值的判断：**对给出的测试集中对应的用户-电影进行预测评分，输出每一条预测评分，并与真实评分进行对比，误差计算使用 SSE 误差平方和**。

选做部分提示：进阶版采用 minihash 算法对特征矩阵进行降维处理，从而得到相似度矩阵，注意 minihash 采用 jarcard 方法计算相似度，特征矩阵应为 01 矩阵。因此进阶版的特征矩阵选取采用方式为，**如果该电影存在某特征值，则特征值为 1，不存在则为 0，从而得到 01 特征矩阵**。

进阶部分：

本次大作业的进阶部分是在基础版本完成的基础上大家可以尝试做的部分。进阶部分的主要内容是使用**迷你哈希（MiniHash）算法对协同过滤算法和基于内容推荐算法的相似度计算进行降维**。同学可以把迷你哈希的模块作为一种近似度的计算方式。

协同过滤算法和基于内容推荐算法都会涉及到相似度的计算，迷你哈希算法在牺牲一定准确度的情况下对相似度进行计算，其能够有效的降低维数，尤其是对大规模稀疏 01 矩阵。同学们可以**使用哈希函数或者随机数映射来计算哈希签名**。哈希签名可以计算物品之间的相似度。

最终降维后的维数等于我们定义映射函数的数量，我们设置的映射函数越少，整体计算量就越少，但是准确率就越低。**大家可以分析不同映射函数数量下，最终结果的准确率有什么差别**。

对基于用户的协同过滤推荐算法和基于内容的推荐算法进行推荐效果对比和分析，选做的完成后再进行一次对比分析。

1.3 实验过程

1.3.1 编程思路

（一）基于用户的协同过滤推荐算法

根据上述要求，基于用户的推荐算法的主要思路是：

① 根据给定数据集构造用户-电影效用矩阵；

② 计算每一对用户之间的 pearson 相似度，构造相似度矩阵；

③ 获取给定用户的前 k 个最相近的用户；

④ 依据这 k 个用户的评分情况，对给定用户的所有未评分电影进行评分预测；

⑤ 选取评分最高的前 n 个电影进行推荐；

⑥ 若对给定的测试集（包含 100 条用户-电影评分记录）进行测试，则进一步计算预测评分与实际评分的误差平方和 SSE。

整体的算法流程图如图 1 所示。

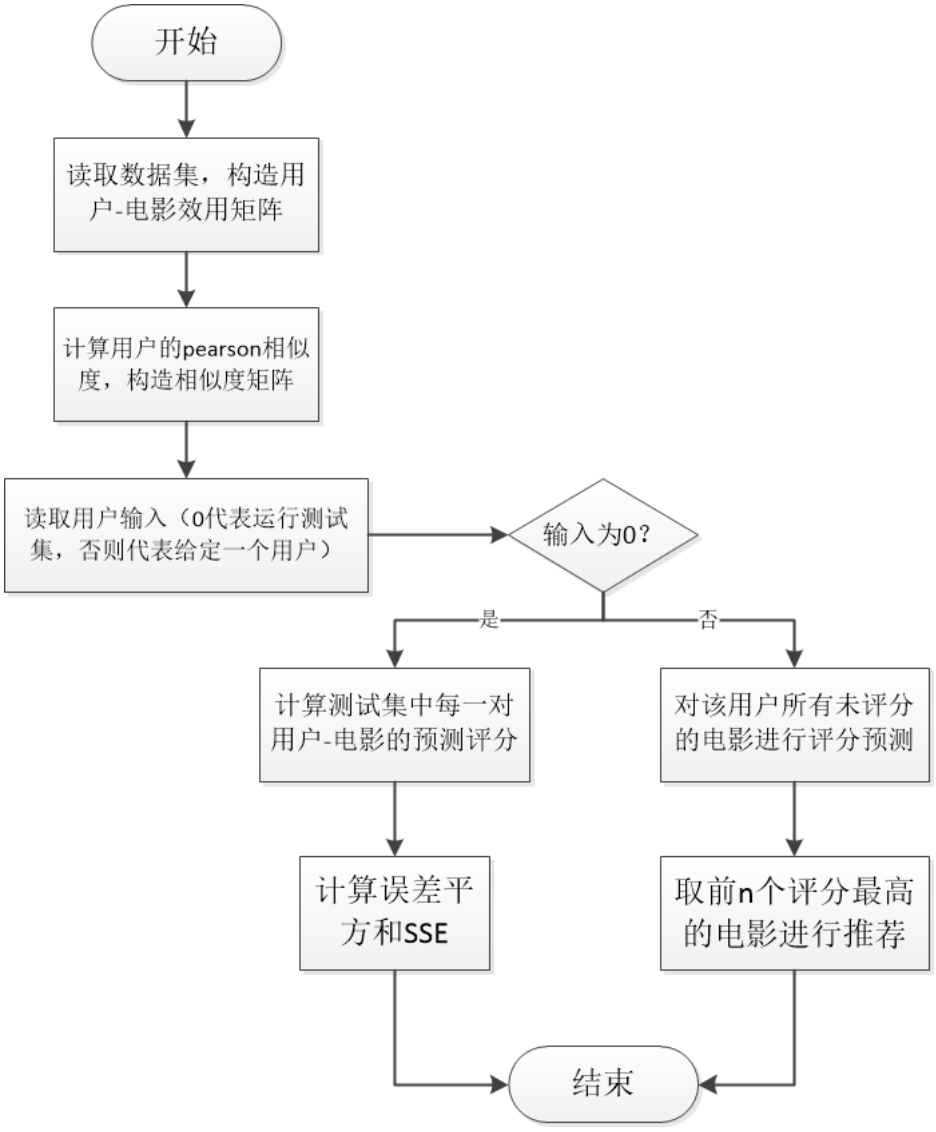

图 1：基于用户的协同过滤推荐算法流程图

下面分别对各个步骤进行介绍。

（1）读入数据集，构造用户-电影效用矩阵，计算每一对用户之间的 pearson 相似度并构造相似度矩阵

基于内容的推荐算法的数据集包括电影信息 movies.csv、用户评分信息 ratings.csv。首先，利用 pandas 的 read_csv 函数读入 movies.csv 中的电影 id（movieId）、电影名称（title）、电影类别（genres），并存储在 movies 和 movies_title 两个数组中。接下来，同样读入 ratings.csv 中的用户 id（userId）、电影 id（movieId）、评分（rating），时间戳（timestamp）不需要读入。两个数据集的预览如图 2 和图 3 所示。

	A	B	C	D	E	F	G
1	movieId	title	genres				
2	1	Toy Story (	Adventure|Animation|Children|Comedy|Fantasy				
3	2	Jumanji (19	Adventure|Children|Fantasy				
4	3	Grumpier (	Comedy|Romance				
5	4	Waiting to	Comedy|Drama|Romance				
6	5	Father of tl	Comedy				
7	6	Heat (1995	Action|Crime|Thriller				
8	7	Sabrina (19	Comedy|Romance				
9	8	Tom and H	Adventure|Children				
10	9	Sudden De	Action				
11	10	GoldenEye	Action|Adventure|Thriller				
12	11	American I	Comedy|Drama|Romance				
13	12	Dracula: D	Comedy|Horror				
14	13	Balto (199	Adventure|Animation|Children				
15	14	Nixon (199	Drama				

图 2：movies.csv 数据集部分预览

	A	B	C	D
1	userId	movieId	rating	timestamp
2	1	31	2.5	1260759144
3	1	1029	3	1260759179
4	1	1061	3	1260759182
5	1	1129	2	1260759185
6	1	1172	4	1260759205
7	1	1263	2	1260759151
8	1	1287	2	1260759187
9	1	1293	2	1260759148
10	1	1339	3.5	1260759125

图 3：ratings.csv 数据集部分预览

读取后统计全部的用户和电影信息，生成一个以用户为行、评分为列的矩阵，每一行代表一个用户对全部电影的评分情况。

最后，只需使用 numpy 的 corrcoef 函数计算用户两两之间的 pearson 相似度，

即可构建相似度矩阵。实现以上功能的代码如下：

```
user_count = 671
def getMovies():
    data = pd.read_csv('movies.csv')
    col_1 = data['movieId']
    col_2 = data['title']
    col_3 = data['genres']
    movies = {}
    movies_title = {}
    i = 0
    for line in col_3:
        arr = line.split("|")
        movies[col_1[i]] = arr
        i += 1
    i = 0
    for line in col_2:
        arr = line
        movies_title[col_1[i]] = arr
        i += 1
    return movies, movies_title

def get_rating():
    f = open('train_set.csv')
    ratings = f.readlines()
    f.close()
    r = []
    ratings.pop(0)
    for line in ratings:
        rate = line.strip().split(',')
        r.append([int(rate[0]), int(rate[1]), float(rate[2])])
    movies = []
    for x in r:
        if x[1] not in movies:
            movies.append(x[1])
    m = len(movies)
    user_movie = np.zeros([user_count, m])
    for item in r:
        y = movies.index(item[1])
        user_movie[item[0]-1, y] = item[2]  # 生成一个以用户为行、评分为
列的矩阵
    user_user = np.corrcoef(user_movie)   # 计算用户的 pearson 相似度矩阵
    return r, user_user
```

上面的 user_user 矩阵用于存放用户间的 pearson 相似度。其计算公式如下：

$$Pearson(x,y)=\frac{n\sum x_i y_i-\sum x_i\sum y_i}{\sqrt{[n\sum {x_i}^2-(\sum x_i)^2][n\sum {y_i}^2-(\sum y_i)^2]}}$$

其中 n 代表电影的总数量，x_i 和 y_i 代表用户 x 和用户 y 对电影 i 的评分。若完全按此公式进行计算，需要设计多重循环遍历统计 $x_i y_i$、x_i、y_i、${x_i}^2$与${y_i}^2$的和，效率较低。因此，我采用了 numpy 自带的库函数 corrcoef 进行计算。

（2）获取给定用户的前 k 个最相近的用户

上一步已经计算出了用户之间的相似度矩阵。对于给定的用户，只需要对其所在列的用户相似度从高到低进行排序，并取前 k 个用户即可。最终将排序结果输出到一个列表 neighbors_dist 中。核心代码如下：

```
def nearuser_k(userid, user_rate, movie_user, user_user):
    neighbors = []
    neighbors_dist = []
    for item in user_rate[userid]:
        for neighbor in movie_user[item[0]]:
            if neighbor != userid and neighbor not in neighbors:
                neighbors.append(neighbor)
                dist = user_user[userid - 1, neighbor - 1]
                neighbors_dist.append([dist, neighbor])
    neighbors_dist.sort(reverse=True)
    return neighbors_dist
```

（3）对给定用户的所有未评分电影进行评分预测

预测评分的计算公式是：

$$\text{predict\_rating}=\frac{\sum_{i=1}^{k} rating(i)*sim(i)}{\sum_{i=1}^{k} sim(i)}$$

其中$rating(i)$代表第i个用户对某一部电影的真实评分，$sim(i)$代表第i个用户与给定用户的 pearson 相似度。

在上一步，已经得到了前 k 个最相近的用户。对于给定用户的每一部未评分的电影，首先按照以上公式加权计算预测评分。一般情况下，不会出现分子部分等于 0 的情况。若分子等于 0，说明这 k 个用户中没有任何一个用户对这部电影评分过。因此，可以取该用户已打分电影的评分平均值作为对这部电影的预期评分。实现评分预测的核心代码如下：

```
def predict_score(userid, movieid, user_rate, movie_user,
user_user, k):
    neighbors_dist = nearuser_k(userid, user_rate, movie_user,
user_user)
    neighbors_dist = neighbors_dist[:k]
```

```
    sum = 0
    for movie in user_rate[userid]:
        sum += movie[1]
    user_acc = sum / len(user_rate[userid])
    sum2 = 0    # 预测评分
    sum3 = 0    # 相似度之和
    for neighbor in neighbors_dist:
        sum1 = 0
        movies = user_rate[neighbor[1]]  # 相似用户对电影的评分列表
        # 计算每一部电影对用户的推荐程度大小
        for movie in movies:
            if movie[0] == movieid:
                sum1 += neighbor[0]                 # 相似度
                sum2 += neighbor[0] * movie[1]      # 相似度*评分
        if sum1 == 0:
            sum1 = neighbor[0]
            sum2 += neighbor[0] * user_acc   # 当所有相似用户都未对该电影进行评分时，认为用户对其评分为其评分的平均值
        sum3 += sum1
    pred_score = sum2 / sum3
    return pred_score
```

（4）选取评分最高的前 n 个电影进行推荐

预测完所有该用户未评分的电影后，只需要对每一部预测出来的电影的评分由高到底进行排序，取前 n 个电影推荐（n 由用户给定），并返回推荐列表 recommend_list 即可。排序的代码如下：

```
recommend_list = []
for key in recommend_dict:
    recommend_rating[key] = recommend_movie[key]/recommend_dict[key]
    recommend_list.append([recommend_dict[key], key])
recommend_list.sort(reverse=True)  # 根据预测评分进行排序
```

（5）运行测试集并计算 SSE

对测试集中的每一条用户 id 和电影 id 进行评分预测，其思路与对上述给定用户推荐电影一致。只需要读取测试集 test_set.csv，调用评分预测函数 predict_score 对所有的用户及对应的电影进行预测并计算误差平方即可。最终将所有误差平方累加得到误差平方和 SSE，其计算公式是：

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

其中 n 是测试集的总项目数，$n = 100$；y_i是测试集中一项的真实评分，$\hat{y}_i$是

该项的预测评分。

（二）基于用户的协同过滤推荐算法进阶部分：Minihash 算法与效用矩阵 0-1 化

Minihash 算法是一种用来快速估算两个集合相似度的哈希算法，可以在牺牲一定准确度的情况下对原始数据进行降维，对稀疏的 0-1 矩阵表现出很好的性能。传统的哈希算法只会将原始内容尽量均匀随机地映射为一个签名值，例如若有两个哈希函数产生了两个签名，且哈希签名相等，则说明原始内容在一定概率下是相等的；如果不相等，只能说明原始内容不相等，而不能表示出原始内容的差异度有多大[4]。

在本实验中，可以将用户-电影的评分效用矩阵作 0-1 处理，并依据 minihash 算法生成哈希签名矩阵。最后，对哈希签名矩阵采用 jarcard 方法，得到用户之间的相似度矩阵（以此代替基础版本的 pearson 相似度矩阵），再调用第一部分的评分预测函数计算出前 n 个评分最高的电影进行推荐，即可对原始处理进行改进。具体流程介绍如下。

（1）对用户-电影效用矩阵进行 0-1 处理

根据任务书要求，在 0-1 矩阵（binary_ratings）中将 0.5-2.5 的评分置为 0、3.0~5.0 的评分置为 1。代码如下：

```
binary_ratings = {}
for user_id in user_rate:
    binary_ratings[user_id] = {}
    for movie_id in user_rate[user_id]:
        if user_rate[user_id][movie_id] < 3:
            binary_ratings[user_id][movie_id] = 0
        else:
            binary_ratings[user_id][movie_id] = 1
```

（2）计算哈希签名矩阵，对原始数据集进行降维

本次实验我最终选定降维到 15 维，由此需要 15 个不同的哈希函数生成 15 个不同的哈希签名，其中第 i 个哈希函数是：

$$h(i) = [(i-1)movie\_id - 1]\ mod\ 19，\ 1 \le i \le 15$$

对某个用户所代表的一行数据，若对某电影的评分为 1，则利用第 i 个哈希函数计算出一个哈希值 $h(i)$。这一个哈希函数对该用户评分为 1 的所有电影的哈希值的最小值作为这个哈希函数产生的签名，最终所有的 15 个签名将构成该用户的哈希签名矩阵。核心代码如下：

```
def minhash_signature(user_rate):
    signature = np.full(hash_num, np.inf)
```

```
    for i in range(hash_num):
        for movie_id in user_rate.keys():
            for user_id in binary_ratings:
                minhashvalue = 999999
                if binary_ratings[movie_id] == 1:
                    temphashvalue = ((i-1)*movie_id-1) % 19  # 哈希函数
                    if temphashvalue < minhashvalue:
                        minhashvalue = temphashvalue
                    signature[i] = minhashvalue
    return signature
```

（3）采用 jarcard 方法计算相似度矩阵

Jarcard 方法的定义是：给定两个集合 A、B，jarcard 相似度的计算公式如下。

$$sim_{jarcard} = \frac{|A \cap B|}{|A \cup B|}$$

其中分子、分母分别代表 A 和 B 的交集与并集的元素数量。根据这一公式，计算 jarcard 相似度的代码是：

```
def jaccard_similarity(s1, s2):
    intersection = len(s1 & s2)
    union = len(s1 | s2)
    if union == 0:
        return 0
    return float(intersection) / union
```

（4）对给定用户的所有未评分电影进行评分预测，选取评分最高的前 n 个电影进行推荐或运行测试集并计算 SSE

这部分操作与基础版本完全一致，不再赘述。

综上所述，基于用户的协同滤波推荐算法（Minihash）的算法流程图如图 4 所示。

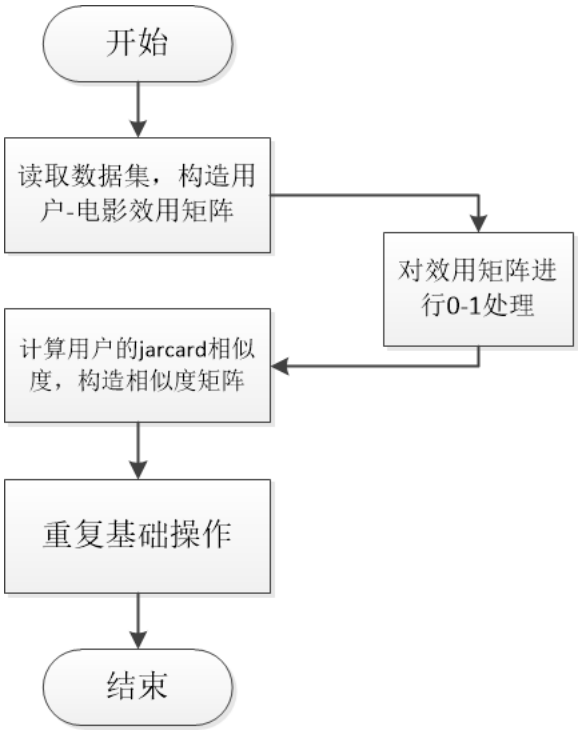

图 4：基于用户的协同过滤推荐算法（Minihash）流程图

（三）基于内容的推荐算法

基于内容的推荐算法主要是根据电影的类别关键词（即标签）进行推荐。其基本思路是分析不同电影之间的相似度，并根据给定用户的已评分电影预测其尚未评分的其它电影的评分。具体流程介绍如下：

（1）读入数据集，构造 TF-IDF 特征矩阵

首先，将数据集 movies.csv 中的电影类别标签作为特征值，并构造 TF-IDF 特征矩阵。设总共有 n 部电影、m 个特征值，最终的特征矩阵的大小为 n×m。对于一部电影，若包含某个特征值，则该位置上的值为 1，反之则为 0。构造特征矩阵的代码如下：

```
a = 0
for i, item in movie_info.items():
    for tag in item[2]:
        b = tags_list.index(tag)
        tf_matrix[a, b] = 1
        idf_matrix[a, b] = 1        #统计所有出现的类别关键词
    a = a + 1
```

接下来分别计算 TF 和 IDF。在信息检索中，TF-IDF 是一种数字统计方法，旨在反映单词对集合或语料库中的文档的重要程度[2]。TF（TermFrequency）为词

频，其值为关键词在全部电影中出现的次数之和与电影中关键词总数的比值；IDF（InverseDocumentFrequency）为逆文档频率。二者的计算公式是：

$$TF = \frac{\text{关键词在全部电影中出现的次数之和}}{\text{关键词总数}}$$

$$IDF = lg\left(\frac{\text{电影总数}}{\text{包含该关键词的电影数} + 1}\right)$$

上面 lg 内分母加 1 是为了避免不存在任何电影包含某关键词的情形，但在此实际上并不会出现这个问题，因为所有关键词都是来自于某些电影的。TF-IDF 值等于 TF 与 IDF 两个值的乘积。

$$TF - IDF = TF \times IDF$$

计算 TF、IDF、TF-IDF 值的代码如下。

```
for j in range(movie_num):
    sum_of_row = sum(tf_matrix[j, :])
    for k in range(tag_num):
        if tf_matrix[j, k]:
            tf_matrix[j, k] = tf_matrix[j, k] / sum_of_row  # 计算 TF（词频）矩阵。词频=词在文件中出现次数/文件中词总数
for j in range(tag_num):
    sum_of_col = sum(idf_matrix[:, j])
    for k in range(movie_num):
        if idf_matrix[k, j]:  # 计算 IDF（反文档频率）。IDF=log(文档总数/包含词的文档总数+1)
            idf_matrix[k, j] = math.log(movie_num / (sum_of_col + 1))
# 其中+1 是为了防止分母为 0
for k in range(movie_num):
    for j in range(tag_num):
        tf_idf[k, j] = idf_matrix[k, j] * tf_matrix[k, j]# 计算 TF*IDF
return tf_idf
```

（2）计算电影之间的相似度并构造相似度矩阵

此处采用余弦相似度的方法计算电影的相似度矩阵。建立一个 n×n 的矩阵保存电影两两之间的 cosine 相似度（余弦相似度），根据上一步计算得到的 TF-IDF 值导出余弦相似度。其计算公式是：

$$sim\_cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}|\cdot|\vec{y}|}$$

（3）对给定用户的所有未评分电影进行评分预测

对每一部用户未评分的电影，获取当前用户的已评分电影，通过上一步得到

的电影余弦相似度矩阵，获得已打分电影与当前预测电影的相似度，并按照下面的方式进行打分计算。选取相似度$sim_{cosine} > 0$的电影进行计算，相似度为负数的电影强制设置为 0，表示其与目标电影无相关性。若分母为 0，则取用户已打分电影的平均评分作为预测评分。

$$\text{score} = \frac{\sum_{i=1}^{n} score'(i) * sim\_cos(i)}{\sum_{i=1}^{n} sim\_cos(i)}$$

其中，n 为计算集合中电影总数，score 为预测评分，score'(i)为计算集合中第 i 个电影的评分，sim_cos(i)为第 i 个电影与当前用户-电影的相似度。核心代码是：

```
def recommendation(user_rate, user_id, movie_ID, cos_sim,
user_movie):
    recommend_list = []
    recommend_dict = {}
    user_rated = user_rate[user_id]
    user_rated_num = len(user_rated)
    movie_num = len(movie_ID)
    for i in range(movie_num):
        if(movie_ID[i]) not in user_movie[user_id]:
            sum1 = 0
            sum2 = 0
            sum3 = 0
            for rated_movie in user_rated:
                row = movie_ID.index(rated_movie[0])
                if cos_sim[row, i] > 0:
                    if movie_ID[i] in user_rated:
                        continue
                    else:
                        sum1 += cos_sim[row, i] * rated_movie[1]
                        sum2 += cos_sim[row, i]
                        sum3 += rated_movie[1]
                else:
                    continue
            if sum2 == 0:
                pre_score = sum3 / user_rated_num   # 取已打分电影的平均值
            else:
                pre_score = sum1 / sum2             # 代入公式计算预期评分
            recommend_list.append([pre_score, i])
            recommend_dict[movie_ID[i]] = pre_score
    recommend_list.sort(reverse=True)
    return recommend_list, recommend_dict
```

（4）选取评分最高的前 n 个电影进行推荐

同理，只需将预测评分列表降序排序，取前 n 个电影推荐即可。

（5）运行测试集并计算 SSE

只需要读取测试集 test_set.csv，调用评分预测函数 predict_score 对所有的用户及对应的电影进行预测并计算误差平方即可。最终将所有误差平方累加得到误差平方和 SSE，其计算公式是：

$$\mathrm{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

其中 n 是测试集的总项目数，$n = 100$；y_i是测试集中一项的真实评分，$\hat{y}_i$是该项的预测评分。

整体的算法流程图如图 5 所示。

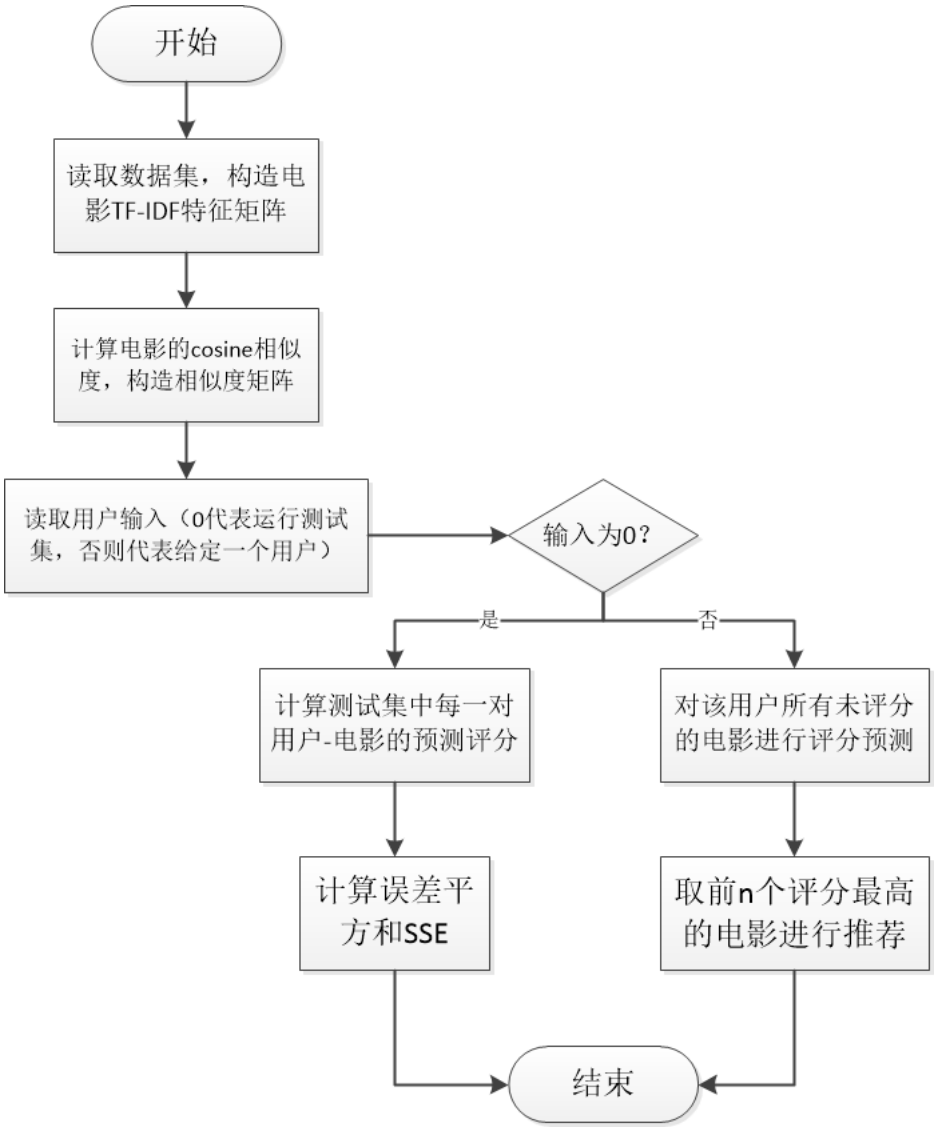

图 5：基于内容的推荐算法流程图

（四）基于内容的推荐算法进阶部分：Minihash 算法与特征矩阵

在本实验中，可以将特征矩阵作 0-1 处理后降维，依据 minihash 算法生成哈希签名矩阵。对哈希签名矩阵采用 jarcard 方法，得到电影之间的相似度矩阵，以此代替基础版本的余弦相似度矩阵，再调用第三部分的评分预测函数计算出前 n 个评分最高的电影进行推荐，即可对原始处理进行改进。具体流程介绍如下。

（1）对特征矩阵进行 0-1 处理

根据任务书要求，在特征矩阵（feature_matrix）中，如果该电影存在某特征值，则特征值为 1，不存在则为 0。代码如下：

```
feature_matrix = np.zeros(movie_info, tags_list)
for i, row in movie_info.iterrows():
    for genre in row['genres'].split('|'):
        feature_matrix[i][tags_list.index(genre)] = 1    # 构造特征矩阵
```

（2）计算哈希签名矩阵，对原始数据集进行降维

本次实验我最终选定降维到 5 维，由此需要 5 个不同的哈希函数生成 5 个不同的哈希签名，其中第 i 个哈希函数是：

$$h(i) = [(i-1)feature\_id - 1]\ mod\ 19，\ 1 \le i \le 5$$

对某个用户所代表的一行数据，若对某关键词的特征值为 1，则利用第 i 个哈希函数计算出一个哈希值 $h(i)$。这一个哈希函数对该用户特征值为 1 的所有关键词的哈希值的最小值作为这个哈希函数产生的签名，最终所有的 5 个签名将构成该电影的哈希签名矩阵。核心代码如下：

```
hash_num = 5          # 降维到 5 维，使用 5 个不同的哈希函数
def minhash_signature(feature_matrix):
    signature = np.full(hash_num, np.inf)
    for i in range(hash_num):
        for feature_id in feature_matrix.keys():
            minhashvalue = 999999
            if user_rate[movie_id] == 1:
                temphashvalue = ((i-1)*feature_id-1) % 19     # 哈希函数
                if temphashvalue < minhashvalue:
                    minhashvalue = temphashvalue
                signature[i] = minhashvalue
    return signature
```

（3）采用 jarcard 方法计算相似度矩阵

此处采用 jarcard 方法计算电影两两之间哈希签名矩阵的相似度，与基于用户的协同过滤推荐算法 Minihash 改进中所用到的 jarcard 方法一致。

（4）对给定用户的所有未评分电影进行评分预测，选取评分最高的前 n 个电影进行推荐或运行测试集并计算 SSE

这部分操作与基础版本完全一致，不再赘述。

综上所述，基于用户的协同滤波推荐算法（Minihash）的算法流程图如图 6 所示。

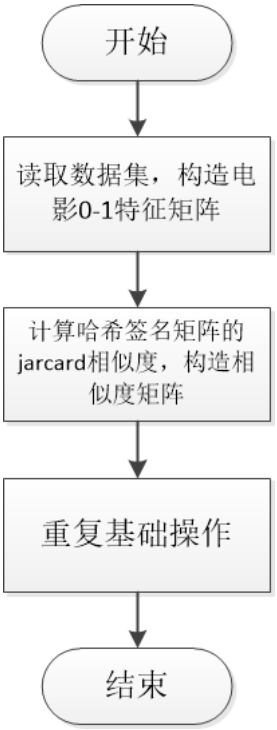

图 6：基于内容的推荐算法（Minihash）流程图

1.3.2 函数模块定义

（一）基于用户的协同过滤推荐算法（含 Minihash）

（1）getMovies

①函数参数：无

②函数功能：读取电影数据集 movies.csv，获得电影 id、电影名称和电影类别标签。

（2）get_rating

①函数参数：无

②函数功能：读取评分数据集 train_set.csv，获得用户 id、电影 id 和评分，生成一个以用户为行、评分为列的矩阵，并计算用户之间的 pearson 相似度矩阵。

（3）get_user

①函数参数：r

②函数功能：根据读入的电影和评分数据集，建立用户字典和电影字典。其中，

用户字典的形式是：user_rate[用户 id]=[(电影 id, 电影评分)...]；电影字典的形式是：movie_user[电影 id]=[用户 id1, 用户 id2...]。

（4）nearuser_k

①函数参数：userid, user_rate, movie_user, user_user

②函数功能：计算与给定用户最相似的 k 个用户，输出相似度从高到低排序的用户列表。

（5）predict_score

①函数参数：userid, movieid, user_rate, movie_user, user_user, k

②函数功能：根据与给定用户最相似的 k 个用户，计算每一部给定用户未评分电影的预测评分。

（6）recommendation

①函数参数：userid, user_rate, movie_user, user_user, k

②函数功能：对所有电影的预测评分从高到低进行排序，并输出推荐电影列表。

（7）jaccard_similarity

①函数参数：s1, s2

②函数功能：计算 0-1 化后的用户-电影效用矩阵每两行之间的 jaccard 相似度。

（8）minhash_signature

①函数参数：user_rate

②函数功能：对一个哈希函数，计算出相应的哈希签名。

（二）基于内容的推荐算法（含 Minihash）

（1）get_movie_info

①函数参数：无

②函数功能：根据读入的电影数据集，建立电影信息数组 movie_info，其结构为 movie_info[电影排序]=[(电影 id, 电影名称, 电影标签)]。

（2）TF-IDF

①函数参数：movie_info

②函数功能：分别计算每部电影在所有特征值上的 TF、IDF、TF-IDF 值，并将结果分别保存在 tf_matrix、idf_matrix、tf_idf 三个矩阵中。

（3）minhash_signature

①函数参数：feature_matrix

②函数功能：根据 0-1 化后的特征矩阵，计算出每种电影全部维的哈希签名。

1.3.3 数据结构与存储形式

（1）movies

①类型：一维数组

②作用：存储从电影数据集 movies.csv 中读入的电影类别标签。

（2）movies_title

①类型：一维数组

②作用：存储从电影数据集 movies.csv 中读入的电影名称。

（3）r

①类型：列表

②作用：存储用户的评分信息。

（4）user_movie

①类型：n×m 的数组（其中 n 为用户总数，m 为电影总数）

②作用：是一个以用户为行、评分为列的稀疏矩阵，存储每个用户对不同电影的评分情况。

（5）user_user

①类型：n×n 的数组（其中 n 为用户总数）

②作用：存储由 user_movie 计算得出的用户两两之间的 pearson 相似度。

（6）user_rate

①类型：字典

②作用：存储每个用户的评分情况，存储形式为 user_rate[用户 id]=[(电影 id, 电影评分)...]。

（7）movie_user

①类型：字典

②作用：存储每个电影被哪些用户评分过，存储形式为 movie_user[电影 id]=[用户 id1, 用户 id2...]。

（8）neighbors_dist

①类型：列表

②作用：存储将相似度由高到低排序的其他用户列表。

（9）recommend_dict

①类型：字典

②作用：存储每一部给定用户未评分的电影的相似度之和。

（10）recommend_movie

①类型：字典

②作用：存储每一部给定用户未评分的电影的评分与相似度的乘积之和，以便于求该电影的加权预测评分。

（11）recommend_rating

①类型：字典

②作用：存储每一部给定用户未评分的电影的预测评分。

（12）recommend_list

①类型：列表

②作用：存储将预测评分从高到低排序的全部未评分电影。

（13）signature（或 minhash_sig）

①类型：h×n 的矩阵（其中 h 是哈希函数的个数，即降维数量；n 为用户总数或电影总数）

②作用：存储哈希签名矩阵，实现原始数据集的降维效果。

（14）binary_ratings

①类型：二维数组

②作用：对用户-电影效用矩阵进行 0-1 处理，若用户 user_id 对电影 movie_id 打分范围在 3.0-5.0 之间，则 binary_ratings[user_id][movie_id]=1，反之则为 0。

（15）tags（或 tags_list）

①类型：数组

②作用：存储每部电影的内容特征标签。

（16）tf_matrix、idf_matrix、tf_idf

①类型：矩阵

②作用：分别存储 TF、IDF、TF-IDF 值。

（17）feature_matrix

①类型：m×f 的矩阵（其中 m 为电影总数，f 为特征总数）

②作用：特征矩阵，若某电影包含某特征关键词，则其位置上的值为 1，反之则为 0。

1.4 遇到的问题与挑战

本次实验主要遇到了下面几个问题。

第一是在做 minhash 改进处理时，不知道哈希函数怎么样设置才能得到较好的降维效果，后来经过反复尝试发现哈希函数模质数 29 时得到的值较为均匀，降维后计算得出的 SSE 也不至于太大。

第二是在计算 pearson 相似度和 cosine 余弦相似度时，一开始想自己手写函数去计算，结果发现 pearson 相似度需要两重循环统计 5 种变量的和，余弦相似度也需要两重遍历两个特征向量，耗时很长。后来直接调用 numpy 的库函数 corrcoef 即可快速计算 pearson 相似度，调用 sklearn.metrics.pairwise 的库函数 cosine_similarity 即可快速计算余弦相似度，大大改善程序运行效率。

第三是在最初计算电影预测评分时，没有考虑到其余相似用户中没有人给某部电影评分的情形，导致出现了图 7 的除 0 错误。

```
ZeroDivisionError: division by zero

Process finished with exit code 1
```

图 7：除 0 错误

对于这种情况，我将给定用户已评分的电影的评分平均值作为预测评分，就解决了这个问题。

1.5 测试结果与分析

1.5.1 测试环境

本次实验的测试环境是：Windows 11 操作系统下的 PyCharm 2023.1 (Community Edition), Build #PC-231.8109.197, built on March 29, 2023, VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

1.5.2 测试结果

（一）基于用户的协同过滤推荐算法

运行“推荐系统_用户.py”，首先测试基础功能。

程序首先输出用户之间的 pearson 相似度矩阵，如图 8 所示。

```
[[ 1.         -0.00396408 -0.00327984 ...  0.06034856 -0.00248654
   0.01263753]
 [-0.00396408  1.          0.11856659 ...  0.01878096  0.16656437
   0.10441969]
 [-0.00327984  0.11856659  1.         ...  0.07680696  0.1330757
   0.16351199]
 ...
 [ 0.06034856  0.01878096  0.07680696 ...  1.          0.03930621
   0.07901782]
 [-0.00248654  0.16656437  0.1330757  ...  0.03930621  1.
   0.22423333]
 [ 0.01263753  0.10441969  0.16351199 ...  0.07901782  0.22423333
   1.        ]]
```

图 8：用户之间的 pearson 相似度矩阵

接下来程序提示“请输入被推荐用户的 id（0-运行测试集）”，输入 0，程序计算并输出测试集中 100 条项目的预测评分和误差平方，最后输出误差平方和 SSE，如图 9、图 10 所示。

```
请输入被推荐用户的id（0-运行测试集）:0
用户id    电影id    预测评分        真实评分        误差平方
   547    1        3.591161    3.500000    0.008310323510280866
   547    6        3.474839    2.500000    0.9503118162297629
   564    1        3.819158    4.000000    0.032703872773599034
   564    2        3.537862    4.000000    0.21357158994998637
   624    1        3.360918    5.000000    2.686589561725465
   624    2        2.993042    3.000000    4.8410732832613914e-05
    15    1        3.177940    2.000000    1.3875434866489775
    15    2        2.729548    2.000000    0.5322403900754036
    73    1        3.592011    5.000000    1.9824340558650435
    73    2        3.381817    2.500000    0.7776005022646493
   452    1        3.479656    3.500000    0.0004138713931664025
```

图 9：测试集每一条的预测评分与真实评分的误差平方（节选）

```
测试集误差平方和SSE:  63.18688238768363

Process finished with exit code 0
```

图 10：测试集误差平方和 SSE

可见，总的 SSE=63.1868，预测效果较好。

接下来，输入用户 id=3，相似用户数量 k=30，推荐电影数量 n=10，程序输出预测评分最高的前 10 部电影的 id、预测评分、电影名和电影标签，如图 11 所示。

```
请输入被推荐用户的id（0-运行测试集）:3
请输入相似用户数量k：30
请输入推荐电影数量n：10
电影id    预测评分        电影名                                                  电影标签
318     4.773274    Shawshank Redemption, The (1994)                        ['Crime', 'Drama']
296     4.411828    Pulp Fiction (1994)                                     ['Comedy', 'Crime', 'Drama', 'Thriller']
356     4.498992    Forrest Gump (1994)                                     ['Comedy', 'Drama', 'Romance', 'War']
593     4.495770    Silence of the Lambs, The (1991)                        ['Crime', 'Horror', 'Thriller']
110     4.210258    Braveheart (1995)                                       ['Action', 'Drama', 'War']
2028    4.332207    Saving Private Ryan (1998)                              ['Action', 'Drama', 'War']
2571    4.303032    Matrix, The (1999)                                      ['Action', 'Sci-Fi', 'Thriller']
480     3.720279    Jurassic Park (1993)                                    ['Action', 'Adventure', 'Sci-Fi', 'Thriller']
2959    3.787753    Fight Club (1999)                                       ['Action', 'Crime', 'Drama', 'Thriller']
7153    4.488947    Lord of the Rings: The Return of the King, The (2003)   ['Action', 'Adventure', 'Drama', 'Fantasy']
```

图 11：基于用户的推荐系统

最后测试 Minihash 改进部分。在哈希函数个数等于 15（降维到 15 维）的条

件下运行测试集，结果如图 12 所示。可见此时的 SSE=67.9702，比基础版本的 SSE 略高，但没有高太多，因此基本实现了评分预测的功能。

```
199   6     3.188092    4.500000    1.7211026915782241
199   10    3.770980    2.500000    1.6153901604910824
285   1     3.428044    4.000000    0.3271336659127465
285   2     3.156266    3.000000    0.0244190627601367
150   1     3.410981    3.000000    0.1689053824471802
150   2     3.089212    3.000000    0.0079587809441669
测试集误差平方和SSE:  67.97021445801522
```

图 12：降维到 15 维的 Minihash 算法测试集误差平方和 SSE

分别测试降维后最终维数 2~15 的情形，其在测试集上的 SSE 如表 1 所示（图片太多不一一列出）。

表 1：降维到 2~15 维的测试集误差平方和 SSE

维数	测试集 SSE	维数	测试集 SSE
2	228.8082	**9**	99.2874
3	194.5738	**10**	95.9174
4	175.1416	**11**	86.2076
5	153.9965	**12**	81.3550
6	137.2508	**13**	76.2477
7	125.6390	**14**	72.0198
8	110.9382	**15**	67.9702

根据上表，可绘制出测试集误差平方和 SSE 随 Minihash 降维维数（哈希函数个数）的变化趋势图，如图 13 所示。

可见，在最终降维维数等于 15 时，已经基本接近了原始版本无 Minihash 的预测效果；而在降维维数很低的时候，整体计算量比较少，但准确率就很低。

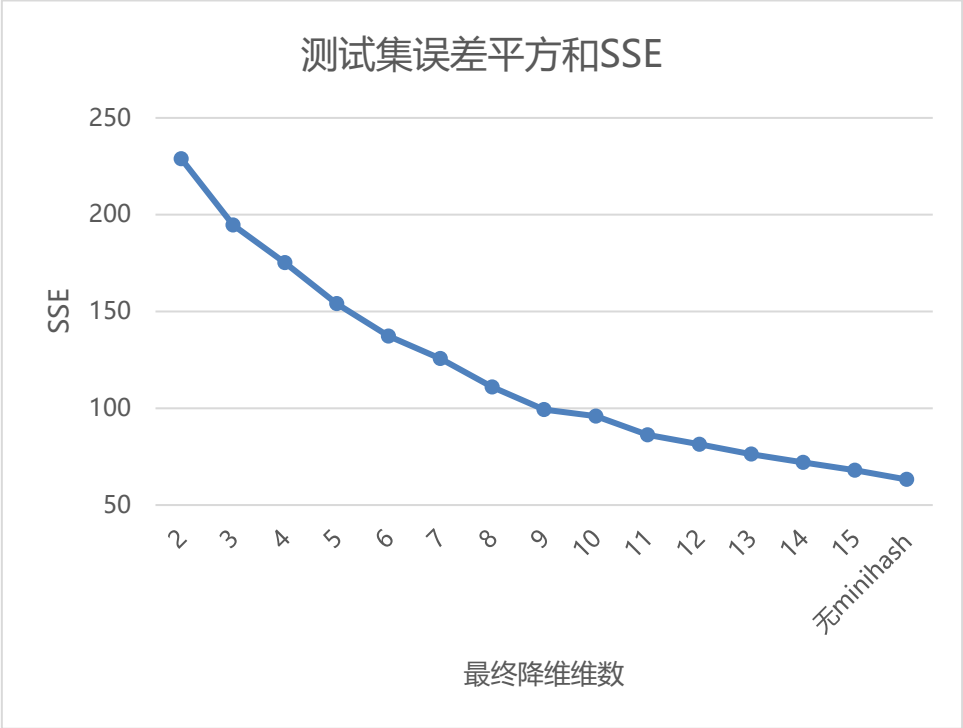

图 13：测试集误差平方和 SSE 随 Minihash 降维维数（哈希函数个数）的变化趋势图

（二）基于内容的推荐算法

运行“推荐系统_内容.py”，首先测试基础功能。

程序首先输出电影的 TF-IDF 矩阵和余弦相似度矩阵，如图 14、图 15 所示。

```
[[0.4198953  0.60279599 0.54977444 ... 0.         0.         0.        ]
 [0.69982551 0.         0.91629073 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

图 14：电影的 TF-IDF 矩阵

```
[[1.         0.80719338 0.09308725 ... 0.         0.18793437 0.        ]
 [0.80719338 1.         0.         ... 0.         0.         0.        ]
 [0.09308725 0.         1.         ... 0.         0.49531786 0.        ]
 ...
 [0.         0.         0.         ... 1.         0.         1.        ]
 [0.18793437 0.         0.49531786 ... 0.         1.         0.        ]
 [0.         0.         0.         ... 1.         0.         1.        ]]
```

图 15：电影的余弦相似度矩阵

接下来程序提示“请输入被推荐用户的 id（0-运行测试集）”，输入 0，程序计算并输出测试集中 100 条项目的预测评分和误差平方，最后输出误差平方和 SSE，如图 16、图 17 所示。

```
请输入被推荐者的id（0-运行测试集）：0
用户id    电影id    预测评分        真实评分        误差平方
   547    1        3.252042    3.500000    0.061482983481165494
   547    6        3.385065    2.500000    0.7833392763861701
   564    1        3.408251    4.000000    0.35016675692249605
   564    2        3.276936    4.000000    0.5228216787699175
   624    1        2.983650    5.000000    4.065666185080809
   624    2        2.970692    3.000000    0.0008589553482891048
    15    1        2.394704    2.000000    0.15579155034739164
    15    2        2.312089    2.000000    0.09739927822233123
    73    1        3.282307    5.000000    2.9504681714095913
    73    2        3.255831    2.500000    0.5712800005870303
```

图 16：测试集每一条的预测评分与真实评分的误差平方（节选）

```
测试集误差平方和SSE= 67.06801578815222

Process finished with exit code 0
```

图 17：测试集误差平方和 SSE

可见，总的 SSE=67.0680，预测效果较好。

接下来，输入用户 id=8，推荐电影数量 n=10，程序输出预测评分最高的前 10 部电影的 id、预测评分、电影名和电影标签，如图 18 所示。

```
请输入被推荐用户的id（0-运行测试集）：8
请输入推荐电影数量n：10
电影id    预期评分        电影名                                                电影标签
  8228    4.376809    Maltese Falcon, The (a.k.a. Dangerous Female) (1931)    ['Mystery']
  2945    4.376809    Mike's Murder (1984)                                    ['Mystery']
 69945    4.294302    Fast and the Furious, The (1955)                        ['Crime', 'Mystery']
 47728    4.294302    Green for Danger (1946)                                 ['Crime', 'Mystery']
  4969    4.294302    And Then There Were None (1945)                         ['Crime', 'Mystery']
  4469    4.294302    Appointment with Death (1988)                           ['Crime', 'Mystery']
  4212    4.294302    Death on the Nile (1978)                                ['Crime', 'Mystery']
108949    4.246481    Art of the Steal, The (2013)                            ['Crime']
 59418    4.246481    American Crime, An (2007)                               ['Crime']
 58904    4.246481    Chan Is Missing (1982)                                  ['Crime']
```

图 18：基于内容的推荐系统

从上图中可以看出，推荐的电影基本都具有相似的类别标签，证明了推荐是以电影内容为依据的。

最后测试 Minihash 改进部分。在哈希函数个数等于 5（降维到 5 维）的条件下运行测试集，误差平方和 SSE=69.1108，比基础版本的 SSE 略高，但没有高太多，因此基本达到了评分预测的功能。

分别测试降维后最终维数 2~5 的情形，其在测试集上的 SSE 如表 2 所示（图片太多不一一列出）。

表 2：降维到 2~5 维的测试集误差平方和 SSE

维数	测试集 SSE	维数	测试集 SSE
2	163.6992	**4**	87.4520
3	111.7275	**5**	69.1108

根据上表，可绘制出测试集误差平方和 SSE 随 Minihash 降维维数（哈希函数个数）的变化趋势图，如图 19 所示。

可见，在最终降维维数等于 5 时，已经基本接近了原始版本无 Minihash 的预测效果；而在降维维数很低的时候，整体计算量比较少，但准确率很低。

图 19：测试集误差平方和 SSE 随 Minihash 降维维数（哈希函数个数）的变化趋势图

1.5.3 结果分析

（一）对原始推荐系统效果的分析

对比基于用户的协同过滤推荐系统与基于内容的推荐系统，发现二者在测试集上的 SSE 值大致相同，表现出的预测效果也基本一致。可见，这两种算法都能有效地针对不同用户推荐合适的电影。

（二）对 Minihash 算法效果的分析

引入 Minihash 后的算法（无论是基于用户的协同过滤推荐系统，还是基于内容的推荐系统）相比原始版本在测试集上的 SSE 都偏大，且程序运行时间略长，推测主要是在计算哈希签名矩阵上耗费了一定时间。实际上，Minihash 在牺牲一定准确度的情况下对相似度进行计算，能有效降低维数，尤其是对大规模稀疏 01 矩阵。

如果哈希函数少，则保留的特征少，最终得到的维度少，降维效果好，但准确率低；反正，如果哈希函数多，则保留的特征多，最终得到的维度多，降维效果差，但准确率高。

（三）二者的比较分析

总体上，基于用户的协同滤波推荐算法在测试集上的性能略好于基于内容的推荐算法。但无论是否引入 Minihash，最终对于给定用户推荐的电影在一定程度上都存在重叠现象，可见二者都能很好地推荐合适的电影。

1.6 心得体会与总结

本学期大数据分析课程的大作业主要聚焦在推荐系统这一主题上，分为基于用户和基于内容两种角度。在具体的实现过程中，这两种算法既有相似之处又有不同之处。相似之处在于，它们都是根据已经得到的历史用户评分数据或电影特征数据，对未评分电影进行预测，并达到推荐的目的；不同之处在于，基于用户的推荐系统主要从用户之间的相似度来做出预测，而基于内容的推荐系统主要关注电影本身的类别特征属性。无论是哪种处理方式，都能产生更符合实际情况的预测结果。

同时，我还发现在一些具体算法的实现中，调用封装好的库函数的效率比自己手写的要好很多，例如在计算 pearson 相似度时，可以对应 numpy 的库函数 corrcoef；计算余弦相似度时，可以选择调用 sklearn.metrics.pairwise 的库函数 cosine_similarity。我认为这可能是由于库函数在很多方面进行了优化，如特殊情形的特判、稀疏矩阵的处理等，因此效率也更高。

通过本次大作业，我进一步掌握了对大规模数据的灵活处理方式，学会了如

何利用多种方式计算相似度，增进了对 Python 编程语言的熟练度，感受到了大数据分析在实际生活中的强大作用与意义。这几次的实验也让我把课堂上所学到的知识与理论进行了真正的实践，提高了编程代码能力，使我受益良多。

1.7 参考文献

[1] TF-IDF 统计方法 https://baike.baidu.com/item/tf-idf/8816134?fr=aladdin
[2] TF-IDF 算法讲解 https://blog.csdn.net/qq_45893319/article/details/119278730
[3] 【机器学习】minHash 最小哈希原理及其应用
https://blog.csdn.net/zfhsfdhdfajhsr/article/details/128529402
[4] MinHash 介绍 https://baike.baidu.com/item/MinHash/7132439?fr=aladdin